04.20.05

Attorney's Docket No.: 07844-315001 / P289

AF

2124

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant	: Gordon B. Dow	Art Unit	: 2126
Serial No.	: 09/293,737	Examiner	: Charles E. Anya
Filed	: April 16, 1999		
Title	: DYNAMIC DEPENDENCY GRAPH IN MVC PARADIGM		

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

SUBMISSION OF APPEAL BRIEF

Further to the Notice of Appeal filed on February 16, 2005 and the Amendment Under 37 CFR § 41.33 filed via facsimile on April 15, 2005, the Applicant submits herewith an Appeal Brief and a check (#187005) in the amount of $500.00 for the Appeal Brief fee.

Respectfully submitted,

Date: 18 Apr 05

Hans R. Troesch
Reg. No. 36,950

Fish & Richardson P.C.
500 Arguello Street, Suite 500
Redwood City, California 94063
Telephone: (650) 839-5070
Facsimile: (650) 839-5071

50272845.doc

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. EV 584 751 552 US

April 18, 2005
Date of Deposit

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant : Gordon B. Dow
Serial No. : 09/293,737
Filed : April 16, 1999
Title : DYNAMIC DEPENDENCY GRAPH IN MVC PARADIGM

Art Unit : 2126
Examiner : Charles Anya

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

BRIEF ON APPEAL

(1) Real Party in Interest

The real party in interest is Adobe Systems Incorporated.

(2) Related Appeals and Interferences

None.

(3) Status of Claims

Claims 19-23, 33, 40, and 54-59 stand rejected under 35 U.S.C. 102(e) as being anticipated by U.S. Pat. No. 6,272,672 B1 to Conway ("Conway").

Claims 1, 2, 5-7, 9, 28-29, 35-36, 42, 45-46, and 48 stand rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Pat. No. 5,469,538 to Razdow ("Razdow") in view of U.S. Pat. No. 5,404,428 to Wu ("Wu").

Claims 11, 13-18, 31-32, 38-39, and 49-53 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Conway in view of Wu.

Claims 3, 4, 8, 43, 44, and 47 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Razdow in view of Wu, and further in view of Conway.

Claims 1-9, 11, 13-23, 28, 29, 31-33, 35, 36, 38-40, and 42-59 are being appealed.

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. EV 584 751 552 US

April 18, 2005
Date of Deposit

(4) Status of Amendments

Amendments to claims 15, 20, 32, 33, 39, 40, and 54 were submitted in an amendment under 37 CFR § 41.33 that was filed on April 15, 2005, to place the claims in better condition for consideration on appeal. The appendix of claims to this brief does not reflect the foregoing amendment.

(5) Summary of Claimed Subject Matter

The claims are directed generally to dependency management among computer program objects representing a state of a computer program application. Specification, page 1, lines 13-14.

Claims 1, 28, and 35 are directed respectively to a method, a system, and a computer program product implementing techniques for maintaining dependencies among a set of objects each having a value. The set of objects includes an object A and an object B. The techniques include marking object A as dirty and not recomputing the value of object A until object A is queried for a value when the value of object A is a function of the value of object B and the value of object B changes. *See* Specification, page 5, lines 11-12, and page 6, lines 9-12. When the value of object B changes, the techniques further include invalidating the dependents of object B and all of their further dependents, including severing dependencies among the dependents of object B and all of their further dependents. *See* Specification, page 5, lines 12-13, and page 6, lines 9-12. The techniques further include causing each invalidated observer-only object to recompute its value by querying the values of the objects from which the observer-only object depends. *See* Specification, page 5, lines 12-14, and page 6, lines 9-14.

Claim 28 includes means-plus-function limitations. The recited functions can be implemented in digital electronic circuitry, or in computer hardware, firmware, software, or in combinations of them. The recited functions can also be implemented in a computer program product tangibly embodied in a machine-readable storage device for execution by a programmable processor. *See* Specification, page 24, lines 4-32, and page 25, lines 1-5.

Claims 7, 29, and 36 are directed respectively to a method, a system, and a computer program product implementing techniques for maintaining dependencies among a set of objects each having a value. The techniques include identifying the objects upon which a given object

depends as those objects into which the given object passed itself as a requester during execution of a compute method of the given object. *See* Specification, page 17, lines 28-30. The techniques further include marking the given object as dirty whenever the value of any one of the identified objects changes and not recomputing the value of the given object until the given object is queried for a value. *See* Specification, page 5, lines 11-12, and page 6, lines 9-12.

Claim 29 includes means-plus-function limitations. The recited functions can be implemented in digital electronic circuitry, or in computer hardware, firmware, software, or in combinations of them. The recited functions can also be implemented in a computer program product tangibly embodied in a machine-readable storage device for execution by a programmable processor. *See* Specification, page 24, lines 4-32, and page 25, lines 1-5.

Claims 11, 31, and 38 are directed respectively to a method, a system, and a computer program product implementing techniques for changing objects having values defining state of a computer program application. The techniques include receiving a change to a value of a changed object. The changed object has objects depending directly on the changed object. The changed object also has objects depending indirectly on the changed object through an object different from the changed object. The changed object is a settable object in the computer program application. *See* Specification, Figures 1A, 1B, 1C, and page 6, lines 15-17. The techniques further include registering the change with a transaction. *See* Specification, Figure 2 and page 6, lines 17-18. The techniques further include dirtying all objects dependent (directly or indirectly) on the changed object. *See* Specification, Figures 1A, 1B, 1C, 2, and page 6, lines 18-20. The techniques further include severing dependencies from the changed object and all of its direct and indirect dependent objects. *See* Specification, Figures 1A, 1B, 1C, 2, and page 6, lines 19-20. Whenever a leaf object is encountered as a dependent object, the techniques further include enqueuing the leaf object for synchronization after the transaction is committed. *See* Specification, Figure 2 and page 6, lines 22-23.

Claim 31 includes means-plus-function limitations. The recited functions can be implemented in digital electronic circuitry, or in computer hardware, firmware, software, or in combinations of them. The recited functions can also be implemented in a computer program product tangibly embodied in a machine-readable storage device for execution by a programmable processor. *See* Specification, page 24, lines 4-32, and page 25, lines 1-5.

Claims 15, 32, and 39 are directed respectively to a method, a system, and a computer program product implementing techniques for changing objects defining state of a computer program application. The techniques include creating a transaction and registering with the transaction one or more changes to settable objects. Each change is made to a corresponding changing object. *See* Specification, Figure 2 and page 6, lines 15-18. For each change registered, the techniques further include traversing a dependency graph from the changing object. *See* Specification, Figure 2 and page 6, lines 18-19. For each dependent object on the dependency graph, the techniques further include marking the dependent object as dirty and detaching the dependent object from the dependency graph. *See* Specification, Figure 2 and page 6, lines 19-20. For each dependent object on the dependency graph, the techniques further include accumulating each leaf object encountered in traversing the dependency graph in a strobe queue. *See* Specification, Figure 2 and page 6, lines 22-25. The techniques further include traversing the strobe queue after all changes to settable objects have been registered. *See* Specification, Figure 2 and page 6, lines 25-30. The techniques further include synchronizing each leaf object by recomputing values for objects marked as dirty and rejoining recomputed objects with the dependency graph, whereby leaf objects are rejoined with the dependency graph. *See* Specification, page 6, lines 15-30.

Claim 32 includes means-plus-function limitations. The recited functions can be implemented in digital electronic circuitry, or in computer hardware, firmware, software, or in combinations of them. The recited functions can also be implemented in a computer program product tangibly embodied in a machine-readable storage device for execution by a programmable processor. *See* Specification, page 24, lines 4-32, and page 25, lines 1-5.

Claims 19 and 40 are directed respectively to a method and a computer program product implementing techniques for managing dependency among a set of objects. Each object of the set has a value. The set includes dependent objects. A given object can have objects depending directly on the given object and objects depending indirectly on the given object through other objects of the set. Each dependent object has a value that is a function of the values of one or more of the other objects in the set. *See* Specification, Figures 1A, 1B, 1C. The techniques include calculating the dependency among objects in the set dynamically at the time objects calculate their values. *See* Specification, page 6, lines 9-14.

Claim 33 is directed to a system for managing dependency among a set of objects. Each object of the set has a value. The set includes dependent objects. Each dependent object has a value that is a function of the values of one or more of the other objects in the set. The system includes means for determining a time at which objects calculate their values. *See* Specification, page 6, lines 11-12. The system further includes means for calculating the dependency among objects in the set dynamically at the time objects calculate their values. *See* Specification, page 6, lines 12-14.

Claim 33 includes means-plus-function limitations. The recited functions can be implemented in digital electronic circuitry, or in computer hardware, firmware, software, or in combinations of them. The recited functions can also be implemented in a computer program product tangibly embodied in a machine-readable storage device for execution by a programmable processor. *See* Specification, page 24, lines 4-32, and page 25, lines 1-5.

(6) Grounds of Rejection to be Reviewed on Appeal

A. Are claims 1, 2, 5-7, 9, 28-29, 35-36, 42, 45-46, and 48 properly rejected under 35 U.S.C. 103(a) as being unpatentable over Razdow in view of Wu?

B. Are claims 11, 13-18, 31-32, 38-39, and 49-53 properly rejected under 35 U.S.C. 103(a) as being unpatentable over Conway in view of Wu?

C. Are claims 19-23, 33, 40, and 54-59 properly rejected under 35 U.S.C. 102(e) as being anticipated Conway?

D. Are claims 3, 4, 8, 43, 44 and 47 properly rejected under 35 U.S.C. 103(a) as being unpatentable over Razdow in view of Wu, and further in view of Conway?

(7) Argument

A. Rejections under 35 U.S.C. 103(a) over Razdow in view of Wu

The Examiner asserts that it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of Wu and Razdow because the teachings of Wu would improve the system of Razdow by optimizing evaluation traversal.

However, the evaluation traversal described in Wu is, in fact, in direct conflict with the goal of the system of Razdow.

The system described in Razdow is directed to a mathematical document editor that can perform live, or real-time, symbolic calculations (column 5, lines 41-45, and column 6, lines 4-9). "When an expression or equation is updated or modified by the user, all equations that include the specially designated symbolic equal sign, together with all related equations to which they are linked, are automatically re-calculated (i.e., symbolically manipulated again) in real-time." In contrast, when data is changed in Wu, recalculation of related data is explicitly avoided at the time of the change (column 9, lines 1-47). This feature of Wu is cited by the Examiner in rejecting claim 7 of the present application. Because the teachings of Wu are disadvantageous and contrary to the goal of Razdow, there is no motivation to combine Wu with Razdow.

For at least this reason, the combination is improper and the rejections based on the combination should be withdrawn.

Moreover, even combined, Razdow and Wu do not teach all of the limitations required by the claims, as will be explained.

Claims 1, 5, 6, 28, 35, 45, and 46

Claim 1 stands rejected as being unpatentable over Razdow in view of Wu. Claim 1 recites a method for maintaining dependencies among a set of objects each having a value. The set of objects includes an object A and an object B. When the value of object A is a function of the value of object B and the value of object B changes, the method includes marking object A as dirty and not recomputing the value of object A until object A is queried for a value. The method further includes severing dependencies among the dependents of object B and all of their further dependents. Neither Razdow nor Wu teaches or suggests severing dependencies among the dependents of object B and all of their further dependents.

The Examiner concedes that Razdow is silent with respect to severing dependencies, but states that Wu teaches severing dependencies among the dependents of object B and all of their further dependents. The Examiner finds this feature in column 9, lines 1-47, in Wu. However, the cited passage actually reads as follows:

> "Each derived item declined in the system in implemented embodiments contains references, in the form of direct function calls for evaluating other items which have validity flags. In this way, when a view model attribute is modified by the application program, derived items dependent upon that attribute change, are invalidated during the change. No calculations are performed at the time of an attribute change. Once the device pipeline requests one of the invalid derived items, then the calculations may be performed. This is performed by descending the acyclic graph until valid item(s) are reached, and then calculating invalid item(s) back up the acyclic graph, clearing invalid flags for the item(s) until the item requested is reached, whose invalid flag is then cleared. The acyclic directed graph describes the derived items that become invalid upon a change in value to any view model attribute. It also describes the dependencies of each derived item on other items, and hence an optimal evaluation traversal for any derived item requested by a pipeline."

As explained in the response to Action of April 8, 2004, the cited passage discloses invalidating dependents, but does not teach or suggest severing dependencies (i.e., relationships between objects). In contrast, as shown in Figures 1B and 1C of the present application, when the value of an object changes, the dependencies between objects can be severed. For instance, when the value of A is -2 (Figure 1B), C only depends on A. When the value of A is 2 (Figure 1C) C depends on both A and B. Accordingly, when the value of A changes from 2 to -2, the dependency between C and B is severed.

In response, the Examiner stated that "invalidating dependencies" is equivalent to "severing dependencies". Even if the Examiner were correct, the cited passage teaches invalidating "derived items" but does not teach invalidating the dependencies (i.e., relationships) between the items. Claim 1 is improperly rejected for at least this reason.

Further as to claim 1, the Examiner states that Razdow teaches marking object A as dirty ("out of date") and not recomputing the value of object A until object A is queried for a value. In an obvious contradiction, however, the Examiner concedes, in reference to claim 7, that Razdow is silent with regard to not recomputing the value of the given object until the given object is queried for value.

Razdow does not teach "not recomputing the value of object A until object A is queried for a value," as required by the claims. In fact, the passage cited by the Examiner in rejecting

claim 1 (column 9, lines 8-26, in Razdow) describes a system in which "each node that has been marked out of date in turn recalculates itself" as the next step after being marked out of date.

The Examiner responded by stating that when the value of object A is a function of the value of object B, as recited in claim 1, object A is "in effect" being queried for value when the value of object B is requested to be modified. The applicant respectfully disagrees, and directs the Board's attention to an example shown in Figures 1B-1C of the present application. In the example, the value of object C is a function of the value of object A, but object C is not queried for value when the value of object A changes. Claim 1 is therefore improperly rejected for at least this additional reason, as well.

The remaining claims 5, 6, 28, 35, 45, 46, and their dependent claims either incorporate the above discussed features of claim 1 or include similar features and, thus, are also improperly rejected for the same reasons set forth above with respect to claim 1.

Claims 2 and 42

Claim 2 stands rejected as unpatentable over Razdow in view of Wu. Claim 2 depends from claim 1 and, thus, is improperly rejected for the same reasons set forth above with respect to claim 1. Claim 2 is further improperly rejected because the art cited by the Examiner in rejecting claim 2 (Figures 3A and 3B, column 8 lines 21-43, column 11, lines 43-67, and column 12, lines 1-41, in Razdow) fails to disclose "providing object B in the construction of object A". It is known to one of ordinary skill in the art that the term "construction", in reference to an object, refers to the creation of an object, where the object is an instance of a class in an object-oriented language. Nothing in Razdow teaches or suggests creating an instance of a class. In fact, there is no mention of object-oriented programming anywhere in Razdow. The cited portions in Razdow at most teach creating variables and structures, but not instances of classes.

Claim 42 incorporates the above discussed features of claim 2 and, thus, is also improperly rejected for the same reasons set forth above with respect to claim 2.

Claims 7, 9, 29, 36, and 48

Claim 7 stands rejected as being unpatentable over Razdow in view of Wu. Claim 7 recites a method for maintaining dependencies among a set of objects each having a value. The

method includes identifying the objects upon which a given object depends as those objects into which the given object passed itself as a requester during execution of a compute method of the given object.

Neither Razdow nor Wu teaches or suggests identifying the objects upon which a given object depends as those objects into which the given object passed itself as a requester during execution of a compute method of the given object.

The Examiner finds this feature in column 8, lines 21-43 and column 11, lines 39-61, in Razdow. However, it is known to one of ordinary skill in the art that an object that has a method is an object that is an instance of a class in an object-oriented language. As explained in reference to claim 2, above, nothing in Razdow teaches or even suggests creating an instance of a class. In fact, there is no mention of object-oriented programming in Razdow at all. Claim 7 is improperly rejected for at least this reason.

Furthermore, claim 7 recites a given object that passes itself to other objects during its compute method. The portions in Razdow cited by the Examiner in rejecting claim 7 describe a numerical dependency graph (Figure 3A, column 8, lines 9-43) and a symbolic dependency graph (Figure 3B, column 11, lines 39-61). Both the numerical dependency graph and the symbolic dependency graph have nodes and arcs. In the numerical dependency graph, the nodes represent programs that calculate numerical values for entered mathematical expressions, and the arcs represent variable names of the corresponding numerical values. In the symbolic dependency graph, the nodes represent mathematical expressions qua expressions, and the arcs represent variable names of the corresponding expressions. However, neither the nodes in Razdow's numerical dependency graph nor the nodes in Razdow's symbolic dependency graph are objects that have methods. Accordingly, a given node in Razdow does not have a compute method, during which the given node passes itself (e.g., as a parameter) to other nodes. Therefore, claim 7 is improperly rejected for at least this additional reason as well.

The remaining claims 9, 29, 36, and 48 either incorporate the above discussed features of claim 7 or include similar features and, thus, are also improperly rejected for the same reasons set forth above with respect to claim 7.

B. Rejections under 35 U.S.C. 103(a) over Conway in view of Wu

The Examiner asserts that it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of Wu and Conway because the teachings of Wu would improve the system of Conway by optimizing evaluation traversal. However, because the evaluation traversal disclosed in Wu is directed to traversing a single acyclic graph that represents dependencies between various items in a coordinate system (claim 1 in Wu), Wu's evaluation traversal is not pertinent to the system in Conway because no such acyclic graph exists in Conway.

Conway describes a network of processing components ("components") through which flow objects can flow (column 3, lines 59-65). The components are interconnected by "wires" (column 14, lines 19-21), and each component can be a dependent of one or more flow objects. Each flow object stores its own "dependent set", and a dependent set of a given flow object is a collection of components that depend on that flow object (column 15, lines 45-67). Therefore, dependencies between components and flow objects in Conway are represented not by a single acyclic graph, but rather by a series of unlinked dependent sets (one for each flow object). Accordingly, the steps of ascending and descending an acyclic graph described in Wu have no relevance to the system in Conway.

For at least this reason, the combination is improper and the rejections based on the combination should be withdrawn.

Moreover, even combined, Conway and Wu do not teach all of the limitations required by the claims, as will be explained.

Claims 11, 13, 31, 38, and 49

Claim 11 stands rejected as being unpatentable over Conway in view of Wu. Claim 11 recites a method for changing objects having values defining state of a computer program application. The method includes receiving a change to a value of a changed object. The method further includes severing dependencies from the changed object and all of its direct and indirect dependent objects. Neither Conway nor Wu teaches or suggests "severing dependencies from the changed object and all of its direct and indirect dependent objects," as required by the claims.

The Examiner asserts that Conway teaches receiving a change to a value of a changed object, the changed object having objects depending directly on the changed object and objects depending indirectly on the changed object through an object different from the changed object. But, as explained in the response to Action of April 8, 2004, Conway does not teach objects depending indirectly on other objects.

Rather, Conway describes a network of processing components ("components") through which flow objects can flow (column 3, lines 59-65). The components are interconnected by "wires" (column 14, lines 19-21), and each component can be a dependent of one or more flow objects (column 15, lines 45-67). However, nothing in Conway teaches or suggests that a component can be a dependent of other components, or that a flow object can be a dependent of other flow objects. Accordingly, dependencies in Conway exist only between components and flow objects, and Conway's "wires" (because they only connect components) do not represent dependencies. Thus, the dependencies between components and flow objects are all direct dependencies because there are no components that depend on flow objects through other components, or through other flow objects.

The Examiner stated that Conway discloses indirect dependencies, citing Figure 6 and column 21, lines 1-46. But that is not the case. The cited figure and passage disclose a coupling protocol illustrated on an example network model that includes components A, B, C, D, and E, and flow objects x and y. There is a direct dependency of component A on flow object x, a direct dependency of component C on flow object x, a direct dependency of component D on flow object x, a direct dependency of component B on flow object y, and a direct dependency of component E on flow object y. Note that nothing in Conway teaches or suggests that a dependent of a given flow object can be a dependent of the "owner" of the given flow object. Therefore, component C does not depend on flow object x through component A, even though A is the owner of object x. Likewise, component D does not depend on flow object x through component A, and component E does not depend on flow object y through component B.

Figure 6 also shows a wire connecting components A and B, a wire connecting components B and C, a wire connecting components B and D, and a wire connecting components B and E. As explained above, however, Conway's wires do not represent dependencies. Therefore, it is not true to say, for example, that since components A and E are

wired through component B, there is an indirect dependency between component E and component A.

Because nothing in Conway teaches or suggests objects depending indirectly on other objects, Conways does not teach or suggest "severing dependencies from the changed object and all of its direct and indirect dependent objects," as recited in claim 11. Further more, Wu does not cure the deficiencies of Conway because, as explained in reference to claim 1, Wu does not teach or suggest severing dependencies. Claim 11 is improperly rejected for at least this reason.

The remaining claims 13, 31, 38, and 49 either incorporate the above discussed features of claim 11 or include similar features and, thus, are also improperly rejected for the same reasons set forth above with respect to claim 11.

Claims 14 and 50

Claim 14 stands rejected as unpatentable over Conway in view of Wu. Claim 14 depends from claim 11 and, thus, is improperly rejected for the same reasons set forth above in reference to claim 11. Claim 14 is further improperly rejected because the art cited by the Examiner in rejecting claim 14 fails to teach "using a requester object to make a transaction consistent, the requester object operating to request an object's value," as recited in claim 14. The Examiner finds this feature in column 21, lines 10-19, in Conway. The cited passage actually states that a "coupling protocol" maintains consistency of data across a wiring diagram. However, Conway's coupling protocol does not include an object operating to request another object's value. Accordingly, Conway does not teach or suggest using a requester object to make a transaction consistent, the requester object operating to request an object's value.

Claim 50 incorporates the above discussed features of claim 14 and, thus, is also improperly rejected for the same reasons set forth above with respect to claim 14.

Claims 15, 17, 18, 32, 39, 52, 53

Claim 15 stands rejected as unpatentable over Conway in View of Wu. Claim 15 recites a method for changing objects defining state of a computer program application. The method includes accumulating each leaf object encountered in traversing a dependency graph in a strobe queue. The method further includes traversing the strobe queue after all changes to settable

objects have been registered and synchronizing each leaf object by recomputing values for objects marked as dirty and rejoining recomputed objects with the dependency graph, whereby leaf objects are rejoined with the dependency graph. Neither Conway nor Wu teaches or suggests traversing a strobe queue after all changes to settable objects have been registered and synchronizing each leaf object by recomputing values for objects marked as dirty and rejoining recomputed objects with the dependency graph, whereby leaf objects are rejoined with the dependency graph, as required by claim 15.

In rejecting claim 11, the Examiner concedes that Conway is silent with reference to enqueuing leaf objects for synchronization. However, the Examiner finds this feature in Wu. Even if the Examiner were correct, nothing in Wu teaches or suggests rejoining leaf objects with the dependency graph.

As explained in reference to claim 1, Wu does not disclose severing dependencies between objects. Therefore, Wu neither teaches detaching objects from a dependency graph, nor rejoining objects with the dependency graph, as recited in claim 15. Accordingly, Wu does not cure the deficiencies of Conway because Wu does not teach or suggest "traversing [a] strobe queue after all changes to settable objects have been registered and synchronizing each leaf object by recomputing values for objects marked as dirty and rejoining recomputed objects with the dependency graph, whereby leaf objects are rejoined with the dependency graph," as recited in claim 15. Claim 15 is improperly rejected for at least this reason.

The remaining claims 17, 18, 32, 39, 52, and 53 either incorporate the above discussed features of claim 15 or include similar features and, thus, are also improperly rejected for the same reasons set forth above with respect to claim 15.

Claims 16 and 51

Claim 16 stands rejected as unpatentable over Conway in view of Wu. Claim 16 depends from claim 15 and, thus, is improperly rejected for the same reasons set forth above in reference to claim 15. Claim 16 is further improperly rejected because the art cited by the Examiner in rejecting claim 16 (column 15, lines 46-67 and column 18, lines 63-67 in Conway) fails to teach a dependency graph that includes intermediate nodes. A dependency graph that includes intermediate nodes includes indirect dependencies (e.g., one object depending on a second object

through a third object). As explained in reference to claim 11, Conway does not disclose a dependency graph that includes indirect dependencies. Accordingly, Conway does not disclose a dependency graph that includes intermediate nodes.

Claim 51 incorporates the above discussed features of claim 16 and, thus, is also improperly rejected for the same reasons set forth above with respect to claim 16.

C. Rejections under 35 U.S.C. 102(e) over Conway

Claims 19, 33, 40, 58, and 59

Claim 19 stands rejected as being anticipated by Conway. Claim 19 recites a method for managing dependency among a set of objects, each object of the set having a value. A given object in the set can have objects depending directly on the given object and objects depending indirectly on the given object through other objects of the set. As explained in reference to claim 11, Conway does not disclose objects that depend indirectly on other objects. Claim 19 is improperly rejected for at least this reason.

Claims 58 and 59 depend from claim 19 and, thus, are also improperly rejected for the same reasons set forth above with respect to claim 19.

Claims 33 and 40, as they would be amended by the amendment filed on April 14, 2005, incorporate the above discussed features of claim 19 and, thus, would also be improperly rejected for the same reasons set forth above with respect to claim 19.

Claims 20 and 54

Claim 20 stands rejected as being anticipated by Conway. Claim 20 depends from claim 19 and, thus, is improperly rejected for the same reasons set forth above in reference to claim 19. Claim 20 is further improperly rejected because Conway fails to disclose a set of objects, where each observed object in the set has one or more accessor methods that return a current value of the observed object, as required by claim 20.

The Examiner finds this feature in column 18, lines 41-67, in Conway. The cited passage describes four message-passing communication acts. Nothing in the cited passage indicates that any of the message-passing communication acts return a current value of an observed object.

Claim 54 incorporates the above discussed features of claim 20 and, thus, is also improperly rejected for the same reasons set forth above with respect to claim 20.

Claims 21 and 55

Claim 21 stands rejected as being anticipated by Conway. Claim 21 depends from claim 19 and, thus, is improperly rejected for the same reasons set forth above in reference to claim 19. Claim 21 is further improperly rejected because Conway fails to disclose a set of objects that includes settable objects, where "each settable object in the set has a value setting method that takes two arguments, namely a transaction argument identifying a transaction with which the change to the settable object's value is registered and a new value for the settable object," as recited in claim 21.

The Examiner finds this feature in column 58, lines 12-46, in Conway. The cited passage describes a "transaction register component" that can be used to change a value associated with an instance of a component (column 7, lines 55-57). However, nothing in the cited passage indicates that the transaction register component, or any other component, has a value setting method that takes two arguments, namely a transaction argument identifying a transaction with which the change to the component's value is registered and a new value for the component.

Claim 55 incorporates the above discussed features of claim 21 and, thus, is also improperly rejected for the same reasons set forth above with respect to claim 21.

Claims 22 and 56

Claim 22 stands rejected as being anticipated by Conway. Claim 22 depends from claim 19 and, thus, is improperly rejected for the same reasons set forth above in reference to claim 19. Claim 22 is further improperly rejected because Conway fails to disclose a set of objects, where "each object in the set descends from a VValue class; each computation operation is represented by a Requester object that is owned by a dependent VValue object, and the Requester object enters the dependent set of one or more VValue objects from which the dependent VValue object depends; and the dependent object uses the Requester object to obtain the object values the dependent object needs to calculate its own value," as recited in claim 22.

Claim 56 incorporates the above discussed features of claim 22 and, thus, is also improperly rejected for the same reasons set forth above with respect to claim 22.

Claims 23 and 57

Claim 23 stands rejected as being anticipated by Conway. Claim 23 depends from claim 19 and, thus, is improperly rejected for the same reasons set forth above in reference to claim 19. Claim 23 is further improperly rejected because Conway fails to disclose "accumulating changes to one or more settable VValue objects in a Transaction object and executing the Transaction object," as recited in claim 23.

The Examiner finds this feature in column 7, lines 41-52, and in column 8, lines 4-35, in Conway. The passage in column 7, lines 41-52, in Conway, describes a "setting component type" that includes a variable that can have a distinct value for each reference to the setting component type. The value associated with each reference can be changed. However, nothing in the passage indicates that anything in the setting component type accumulates changes to one or more settable VValue objects in a Transaction object.

The passage in column 8, lines 4-35, in Conway, describes a method for handling control flow in a network of components. The method includes managing a transaction object using flows and message-passing communication acts. However, nothing in the passage indicates that the transaction object accumulates changes to one or more settable VValue objects.

Claim 57 incorporates the above discussed features of claim 23 and, thus, is also improperly rejected for the same reasons set forth above with respect to claim 23.

D. Rejections under 35 U.S.C. 103(a) over Razdow in view of Wu, and further in view of Conway

As explained above, there is no motivation to combine Razdow with Wu, or to combine Conway with Wu. All the more, the combination Razdow, Wu, and Conway is improper, and the rejections based on the three-way combination should be withdrawn.

Moreover, even the combination of Razdow, Wu, and Conway does not teach all of the limitations required by the claims, as will be explained.

Claims 3, 4, 43, and 44

Claims 3 stands rejected as being unpatentable over Razdow in view of Wu, and further in view of Conway. Claim 3 depends from claim 1 and, thus, is improperly rejected for the same reasons set forth above in reference to claim 1. Claim 3 is further improperly rejected because the art cited by the Examiner fails to disclose "providing in object B a handleRequest method that adds a requester owned by object A to a dependents list for object B, the dependents list identifying all objects whose value is a function of the value of object B," as recited in claim 3.

The Examiner finds this feature in Figure 6 and column 21, lines 25-65, in Conway. The cited passage and figure disclose a coupling protocol illustrated on an example network model that includes components A, B, C, D, and E, and flow objects x and y. However, nothing in the cited passage suggests that components have dependents lists. Although Conway's flow objects have "dependent sets" (Figure 4 and column 15, lines 45-67), this does not teach or suggest that a given flow object has a method to add to its dependent set a requester owned by another flow object, as recited in claim 3.

The remaining claims 4, 43, and 44 incorporate the above discussed features of claim 3 and, thus, are also improperly rejected for the same reasons set forth above with respect to claim 3.

Claims 8 and 47

Claim 8 stands rejected as being unpatentable over Razdow in view of Wu, and further in view of Conway. Claim 8 depends from claim 7 and, thus, is improperly rejected for the same reasons set forth above in reference to claim 7. Claim 8 is further improperly rejected because the art cited by the Examiner fails to disclose objects that pass themselves as requester objects to other objects (e.g., to a root object).

The Examiner finds this feature in Figure 6, in Conway, and in column 21, lines 25-55, in Conway. The cited passage and figure disclose a coupling protocol illustrated on an example network model that includes components A, B, C, D, and E, and flow objects x and y. The steps of the coupling protocol include a number of communication acts, such as sending a "notify owner" message from component C to flow object x, sending a "notify dependents" message

from component A to flow object x, and so on. However, none of the communication acts in the cited passage include objects passing themselves (e.g., as parameters) to other objects.

Claim 47 incorporates the above discussed features of claim 8 and, thus, is also improperly rejected for the same reasons set forth above with respect to claim 8.

The brief fee of $500 is enclosed. Please apply any other charges or credits to Deposit Account No. 06-1050.

Respectfully submitted,

Date: 18 Apr 05

Hans R. Troesch
Reg. No. 36,950

Fish & Richardson P.C.
500 Arguello Street, Suite 500
Redwood City, California 94063
Telephone: (650) 839-5070
Facsimile: (650) 839-5071

Appendix of Claims

1. (Original) In a computer program, a method for maintaining dependencies among a set of objects each having a value, the set of objects including an object A and an object B, the method for maintaining dependencies comprising:

when the value of object A is a function of the value of object B and the value of object B changes, marking object A as dirty and not recomputing the value of object A until object A is queried for a value;

when the value of object B changes, invalidating the dependents of object B and all of their further dependents, including severing dependencies among the dependents of object B and all of their further dependents; and

causing each invalidated observer-only object to recompute its value by querying the values of the objects from which the observer-only object depends.

2. (Original) The method of claim 1, further comprising:

providing object B in the construction of object A, wherein the value of object A is a function of the value of the object B that was provided in the construction of object A.

3. (Original) The method of claim 1, further comprising:

providing in object B a handleRequest method that adds a requester owned by object A to a dependents list for object B, the dependents list identifying all objects whose value is a function of the value of object B.

4. (Original) The method of claim 3, wherein the dependents lists for all objects in the set collectively define a directed, acyclic dependency graph.

5. (Original) The method of claim 1, further comprising:

when an object is marked as dirty, breaking any dependency relationships the marked object may have had; and

when the value of an object is recomputed, identifying the objects on which the recomputed value is actually dependent and identifying the recomputed object as dependent only on the identified objects.

6. (Original) The method of claim 1, wherein the set of objects includes settable objects and dependent objects, and each dependent object maintains a flag whose setting marks the dependent object as valid or invalid (i.e., dirty).

7. (Original) In a computer program, a method for maintaining dependencies among a set of objects each having a value, the method for maintaining dependencies comprising:

identifying the objects upon which a given object depends as those objects into which the given object passed itself as a requester during execution of a compute method of the given object; and

marking the given object as dirty whenever the value of any one of the identified objects changes and not recomputing the value of the given object until the given object is queried for a value.

8. (Original) The method of claim 7, further comprising:

identifying as dependents of a root object all objects that passed themselves as requester objects to the root object or to a dependent of the root object during execution of the requester objects' respective compute methods, whereby the set of dependents of the root object is a set that changes based on the computation of dependents and not the root object itself.

9. (Original) The method of claim 7, wherein the set of objects includes settable objects and dependent objects, and each dependent object maintains a flag whose setting marks the dependent object as valid or invalid (i.e., dirty).

10. (Canceled)

11. (Previously Presented) A method for changing objects having values defining state of a computer program application, comprising:

receiving a change to a value of a changed object, the changed object having objects depending directly on the changed object and objects depending indirectly on the changed object through an object different from the changed object, the changed object being a settable object in the computer program application;

registering the change with a transaction;

dirtying all objects dependent (directly or indirectly) on the changed object;

severing dependencies from the changed object and all of its direct and indirect dependent objects; and

whenever a leaf object is encountered as a dependent object, enqueuing the leaf object for synchronization after the transaction is committed.

12. (Canceled)

13. (Original) The method of claim 11, wherein leaf object synchronization comprises:

recomputing a value for each objects marked as dirty, identifying the objects on which the recomputed value is actually dependent, and identifying the recomputed object as dependent only on the identified objects.

14. (Original) The method of claim 13, further comprising:

using a requester object to make the transaction consistent, the requester object operating to request an object's value so that the requested value cannot change until the requester terminates, at which time all objects whose values were requested by the requester object are released.

15. (Original) A method for changing objects defining state of a computer program application, comprising:

creating a transaction registering with the transaction one or more changes to settable objects, each change being made to a corresponding changing object;

for each change registered, traversing a dependency graph from the changing object and (i) for each dependent object on the dependency graph, marking the dependent object as dirty and detaching the dependent object from the dependency graph, and (ii) accumulating each leaf object encountered in traversing the dependency graph in a strobe queue; and

traversing the strobe queue after all changes to settable objects have been registered and synchronizing each leaf object by recomputing values for objects marked as dirty and rejoining recomputed objects with the dependency graph, whereby leaf objects are rejoined with the dependency graph.

16. (Original) The method of claim 15, wherein:

the dependency graph represents application state;

the roots of the dependency graph are the settable objects of the application state; and

the intermediate nodes of the dependency graph are dependent objects whose values are the results of intermediate computations.

17. (Original) The method of claim 15, wherein:

the leaf objects of the dependency graph are coupled to a user interface.

18. (Original) The method of claim 17, wherein:

the leaf objects are coupled directly to the user interface.

19. (Previously Presented) In a computer program, a method for managing dependency among a set of objects, each object of the set having a value, the set including dependent objects, wherein a given object can have objects depending directly on the given object and objects depending indirectly on the given object through other objects of the set, each dependent object having a value that is a function of the values of one or more of the other objects in the set, the method comprising:

calculating the dependency among objects in the set dynamically at the time objects calculate their values.

20. (Previously Presented) The method of claim 19, wherein each observed object in the set has one or more accessor methods that each take a requester argument and returns a current value of the observed object, the requester argument identifying the object requesting the value of the observed object.

21. (Previously Presented) The method of claim 19, wherein each settable object in the set has a value-setting method that takes two arguments, namely a transaction argument identifying a transaction with which the change to the settable object's value is registered and a new value for the settable object.

22. (Original) The method of claim 19, wherein:

each object in the set descends from a VValue class;

each computation operation is represented by a Requester object that is owned by a dependent VValue object, and the Requester object enters the dependent set of one or more VValue objects from which the dependent VValue object depends; and

the dependent object uses the Requester object to obtain the object values the dependent object needs to calculate its own value.

23. (Original) The method of claim 22, wherein a Transaction class descends from the Requester class, the method further comprising:

accumulating changes to one or more settable VValue objects in a Transaction object; and

executing the Transaction object.

24-27. (Canceled)

28. (Original) A system for maintaining dependencies among a set of objects in a computer program, each object having a value, the set of objects including an object A and an object B, the system comprising:

means for recomputing the value of object A, wherein when the value of object A is a function of the value of object B and the value of object B changes, marking object A as dirty and not recomputing the value of object A until object A is queried for a value;

means for recomputing the value of object B, wherein when the value of object B changes, invalidating the dependents of object B and all of their further dependents, including severing dependencies among the dependents of object B and all of their further dependents; and

means for causing each invalidated observer-only object to recompute its value by querying the values of the objects from which the observer-only object depends.

29. (Original) A system for maintaining dependencies among a set of objects in a computer program, each object having a value, the system comprising:

means for identifying the objects upon which a given object depends as those objects into which the given object passed itself as a requester during execution of a compute method of the given object; and

means for marking the given object as dirty whenever the value of any one of the identified objects changes and not recomputing the value of the given object until the given object is queried for a value.

30. (Canceled)

31. (Previously Presented) A system for changing objects having values defining state of a computer program application, comprising:

means for receiving a change to a value of a changed object, the changed object being a settable object in the computer program application;

means for registering the change with a transaction;

means for dirtying all objects dependent (directly or indirectly) on the changed object; and

means for severing dependencies from the changed object and all of its direct and indirect dependent objects; wherein

whenever a leaf object is encountered as a dependent object, the leaf object is enqueued for synchronization after the transaction is committed.

32. (Original) A system for changing objects defining state of a computer program application, comprising:

means for creating a transaction registering with the transaction one or more changes to settable objects, each change being made to a corresponding changing object;

means for traversing a dependency graph, for each change registered, from the changing object and (i) for each dependent object on the dependency graph, marking the dependent object as dirty and detaching the dependent object from the dependency graph, and (ii) accumulating each leaf object encountered in traversing the dependency graph in a strobe queue; and

means for traversing the strobe queue after all changes to settable objects have been registered and synchronizing each leaf object by recomputing values for objects marked as dirty and rejoining recomputed objects with the dependency graph, whereby leaf objects are rejoined with the dependency graph.

33. (Original) A system for managing dependency among a set of objects in a computer program, each object of the set having a value, the set including dependent objects, each dependent object having a value that is a function of the values of one or more of the other objects in the set, the system comprising:

means for determining a time at which objects calculate their values; and

means for calculating the dependency among objects in the set dynamically at the time objects calculate their values.

34. (Canceled)

35. (Original) A computer program product, tangibly stored on a computer-readable medium, for maintaining dependencies among a set of objects each having a value, the set of

objects including an object A and an object B, the product comprising instructions operable to cause a computer to:

recompute the value of object A, wherein when the value of object A is a function of the value of object B and the value of object B changes, object A is marked as dirty and the value of object A is not recomputed until object A is queried for a value;

recompute the value of object B, wherein when the value of object B changes, the dependents of object B and all of their further dependents are invalidated, and the dependencies among the dependents of object B and all of their further dependents are severed; and

cause each invalidated observer-only object to recompute its value by querying the values of the objects from which the observer-only object depends.

36. (Original) A computer program product, tangibly stored on a computer-readable medium, for maintaining dependencies among a set of objects in a computer program, each object having a value, the product comprising instructions operable to cause a computer to:

identify the objects upon which a given object depends as those objects into which the given object passed itself as a requester during execution of a compute method of the given object; and

mark the given object as dirty whenever the value of any one of the identified objects changes and not recompute the value of the given object until the given object is queried for a value.

37. (Canceled)

38. (Previously Presented) A computer program product, tangibly stored on a computer-readable medium, for changing objects having values defining state of a computer program application, the product comprising instructions operable to cause a computer to:

receive a change to a value of a changed object, the changed object being a settable object in the computer program application;

register the change with a transaction;

dirty all objects dependent (directly or indirectly) on the changed object;

sever dependencies from the changed object and all of its direct and indirect dependent objects; and

whenever a leaf object is encountered as a dependent object, enqueue the leaf object for synchronization after the transaction is committed.

39. (Original) A computer program product, tangibly stored on a computer-readable medium, for changing objects defining state of a computer program application, the product comprising instructions operable to cause a computer to:

create a transaction registering with the transaction one or more changes to settable objects, each change being made to a corresponding changing object;

traverse a dependency graph, for each change registered, from the changing object and (i) for each dependent object on the dependency graph, marking the dependent object as dirty and detaching the dependent object from the dependency graph, and (ii) accumulating each leaf object encountered in traversing the dependency graph in a strobe queue; and

traverse the strobe queue after all changes to settable objects have been registered and synchronizing each leaf object by recomputing values for objects marked as dirty and rejoining recomputed objects with the dependency graph, whereby leaf objects are rejoined with the dependency graph.

40. (Original) A computer program product, tangibly stored on a computer-readable medium, for managing dependency among a set of objects in a computer program, each object of the set having a value, the set including dependent objects, each dependent object having a value that is a function of the values of one or more of the other objects in the set, the product comprising instructions operable to cause a computer to:

calculate the dependency among objects in the set dynamically at the time objects calculate their values.

41. (Canceled)

42. (Previously Presented) The product of claim 35, further comprising instructions operable to:

provide object B in the construction of object A, wherein the value of object A is a function of the value of the object B that was provided in the construction of object A.

43. (Previously Presented) The product of claim 35, further comprising instructions operable to:

provide in object B a handleRequest method that adds a requester owned by object A to a dependents list for object B, the dependents list identifying all objects whose value is a function of the value of object B.

44. (Previously Presented) The product of claim 43, wherein the dependents lists for all objects in the set collectively define a directed, acyclic dependency graph.

45. (Previously Presented) The product of claim 35, further comprising instructions operable to:

break any dependency relationships the marked object may have had when an object is marked as dirty; and

when the value of an object is recomputed, identify the objects on which the recomputed value is actually dependent and identify the recomputed object as dependent only on the identified objects.

46. (Previously Presented) The product of claim 35, wherein the set of objects includes settable objects and dependent objects, and each dependent object maintains a flag whose setting marks the dependent object as valid or invalid (i.e., dirty).

47. (Previously Presented) The product of claim 36, further comprising instructions operable to:

identify as dependents of a root object all objects that passed themselves as requester objects to the root object or to a dependent of the root object during execution of the requester

objects' respective compute methods, whereby the set of dependents of the root object is a set that changes based on the computation of dependents and not the root object itself.

48. (Previously Presented) The product of claim 36, wherein the set of objects includes settable objects and dependent objects, and each dependent object maintains a flag whose setting marks the dependent object as valid or invalid (i.e., dirty).

49. (Previously Presented) The product of claim 38, wherein leaf object synchronization comprises:

recomputing a value for each objects marked as dirty, identifying the objects on which the recomputed value is actually dependent, and identifying the recomputed object as dependent only on the identified objects.

50. (Previously Presented) The product of claim 38, further comprising instructions operable to:

use a requester object to make the transaction consistent, the requester object operating to request an object's value so that the requested value cannot change until the requester terminates, at which time all objects whose values were requested by the requester object are released.

51. (Previously Presented) The product of claim 39, wherein:

the dependency graph represents application state;

the roots of the dependency graph are the settable objects of the application state; and

the intermediate nodes of the dependency graph are dependent objects whose values are the results of intermediate computations.

52. (Previously Presented) The product of claim 39, wherein:

the leaf objects of the dependency graph are coupled to a user interface.

53. (Previously Presented) The product of claim 52, wherein:

the leaf objects are coupled directly to the user interface.

54. (Previously Presented) The product of claim 40, wherein each observed object in the set has one or more accessor methods that each take a requester argument and returns a current value of the observed object, the requester argument identifying the object requesting the value of the observed object.

55. (Previously Presented) The product of claim 40, wherein each settable object in the set has a value-setting method that takes two arguments, namely a transaction argument identifying a transaction with which the change to the settable object's value is registered and a new value for the settable object.

56. (Previously Presented) The product of claim 40, wherein:

each object in the set descends from a VValue class;

each computation operation is represented by a Requester object that is owned by a dependent VValue object, and the Requester object enters the dependent set of one or more VValue objects from which the dependent VValue object depends; and

the dependent object uses the Requester object to obtain the object values the dependent object needs to calculate its own value.

57. (Previously Presented) The product of claim 56, wherein a Transaction class descends from the Requester class, the product further comprising instructions operable to:

accumulate changes to one or more settable VValue objects in a Transaction object; and

execute the Transaction object.

58. (Previously Presented) The method of claim 19, wherein all objects of the set are instantiated from object-oriented programming classes that inherit a set of methods from a common base class.

59. (Previously Presented) The method of claim 58, wherein the common base class is a requester class with methods to lock down and reset queried values in order to guarantee consistency.